FLOATING-POINT DSP BLOCK ARCHITECTURE FOR FPGAS

Martin Langhammer, Bogdan Pasca

Altera European Technology Centre, UK

ABSTRACT

This work describes the architecture of a new FPGA DSP block supporting both fixed and floating point arithmetic. Each DSP block can be configured to provide one single precision IEEE-754 floating multiplier and one IEEE-754 floating point adder, or when configured in fixed point mode, the block is completely backwards compatible with current FPGA DSP blocks. The DSP block operating frequency is similar in both modes, in the region of 500MHz, offering up to 2 GMACs fixed point and 1 GFLOPs performance per block. In floating point mode, support for multi-block vector modes are provided, where multiple blocks can be seamlessly assembled into any size real or complex dot products. By efficient reuse of the fixed point arithmetic modules, as well as the fixed point routing, the floating point features have only minimal power and area impact. We show how these blocks are implemented in a modern Arria 10 FPGA family, offering over 1 TFLOPs using only embedded structures, and how scaling to multiple TFLOPs densities is possible for planned devices.

1. INTRODUCTION

FPGAs are increasingly being used as compute platforms, whether in traditional computing applications, or in more complex embedded functions. Floating-point (FP) arithmetic is required for many of these use cases, which often include algorithms such as matrix decompositions. Mainstream adoption of FPGAs as compute accelerators requires FP capabilities ranging from basic operators to elementary functions. It was shown that FPGAs can outperform competing platforms when computing elementary functions [1, 2]; however, operations such as matrix decompositions require simpler standard operators, such as adders and multipliers, but in great numbers. Implementing these large datapaths requires significant logic resources which impact area, power and latency.

Embedded floating-point units are therefore required, but in the context of an FPGA and not an Application Specific Standard Product (ASSP). To make FPGAs floating-point usable while maintaining the economies of scale of a standard FPGA product, such units must have minimal impact on area, power, and fixed-point functionality. This is one key factor often overlooked in previous works which embedded FP functionality in the FPGA fabric. Many device families from both major vendors [3, 4] have several product groupings, each with different ratios of embedded functions to soft logic, such as transceivers in the case of wireline targeted devices, and DSP blocks in the case of signal processing targeted devices. The cost of the DSP blocks for a given device is determined by percentage of core area, from less than 1% for a transceiver focused device to 5% for a DSP focused device. Therefore, the addition of floating-point functionality to DSP blocks is worthy goal; if a modest area penalty is applied to every DSP block, there will be essentially a negligible impact to non-DSP users.

The problem is in finding an effective DSP architecture that supports both useful fixed and floating-point implementations. Ideally, the fixed-point functionality will be backwards compatible to current DSP blocks, with the floating point features introducing no performance, latency, or power penalty. Conversely, the floating-point features must also offer the same clock rates and power consumption as the fixed point modes. In many DSP or arithmetic algorithms, one multiplier output is added to another, which is well supported by existing DSP block architectures; for example, cascade chains from one DSP block to another to support FIR filters in direct form I or systolic modes. These types of features must also be considered for floating-point DSP blocks, but the different bus width to function densities will require new types of inter-function connection structures.

The contributions of this paper are two-fold: (a) embedding high performance FP capabilities into the DSP blocks by reusing the fixed-point hardware, keeping full fixed-point backwards compatibility and maintaining fixed-point performance, and (b) cascading DSP block structures in FP mode that allow for efficient mapping of scalar-product operations. The efficient reuse of existing fixed-point hardware, combined with the low-area and low-latency design for the FP cores, requires an approximate 10% DSP area overhead to provide a fully functional single-precision floating-point unit (adder and multiplier), supporting rounding to nearest (RNE) and with subnormal flushing to zero on inputs and outputs. This has a maximum of 0.5% silicon penalty for applications not using this feature, even in the case of a DSP targeted device with 5% of total die core area used for

DSP blocks. Embedding FP blocks as separate units, which is suggested by previous works, would lead to significantly larger area penalties in non-FP applications.

The rest of the paper is organized as follows. Section 2 briefly reviews floating point terminology. Section 3 reviews prior work on embedded floating-point functions in FPGAs. Section 4 introduces the new DSP block architecture that supports both fixed and FP arithmetic, as well as discussing the multi-block, or vector, mode. Section 5 gives a more detailed description of the micro-architecture of the arithmetic structures used in the block. Section 6 presents results for common applications, including FFTs, FIR filters, matrix-matrix multiplies, and matrix decompositions. It also includes a qualitative results comparison to previous works. Finally, conclusions and the references are presented.

2. BACKGROUND

The IEEE-754 standard for floating-point arithmetic (revised in 2008) [5] uses a triplet to represent a FP value x: s – sign bit (0 for positive, 1 for negative), e – exponent (an integer value) and m – the significand. The value of x is $x = (-1)^s 2^e m$. In order to avoid duplicate representations of the same value, the standard uses a normalized significand – usually $m \in [1,2)$. This allows writing $x = (-1)^s 2^e 1.f$ where f is a fraction $\in [0,1)$. The number of bits used to represent the exponent (wE) and fraction (wF) define the formats of the standard. For instance single-precision uses wE=8 and wF=23.

Representing values that require an exponent smaller than $e_{\min}$ exponent (for that particular format) may be done by denormalising the significand. The property of denormalizing the significand for numbers very close to zero is often referred to as subnormal support (gradual underflow). Subnormal support is often not available in the embedded context since it adds significant resources to the hardware implementation of basic operators.

The IEEE-754 standard proposes several rounding modes for choosing the point on the grid to use for a specific real value: round to nearest (even, away), towards 0 and towards $+\infty$ or $-\infty$. The round to nearest modes are the most commonly used since they introduce the smallest error.

In this work both the floating point multiplier and adder support IEEE-754 arithmetic, with the following limitations. The only rounding mode supported is round to nearest tie-breaks-to-even (RNE). Subnormals are flushed to zero on both inputs and outputs. All other signalling and exception modes, such as zero, NaN, and infinity are properly handled on input, and correctly encoded on the output, although all NaNs are handled as quiet NaNs. Flags are not currently supported, as we felt traps were unlikely to be needed in large, parallel datapath applications such as those for which this core is optimized.

3. PREVIOUS WORK

Several earlier papers describe adding embedded floating point features to FPGAs [6, 7, 8, 9]. In [7], Beauchamp et.al. compare three alternate methods for implementing efficient floating-point computations in FPGA devices. One of these methods involves embedding a dedicated double-precision floating-point DSP block into the fabric, supporting a floating-point multiply-add operation. The performance and silicon area are estimated from published microprocessor FPU data. They conservatively estimate the FPU to be equivalent in area to 161 Xilinx Virtex 2 CLBs (1288 4-LUTs). The area of a single-precision implementation can be roughly approximated based on the single to double interface (1:2) or gate complexity (1:4) ratio, to 40-80 CLBs (300-600 4-LUTs) using their methodology. The resource savings when using the embedded floating-point block was assessed against traditional logic and multiplier-based implementations on a number of floating point benchmarks (common DSP and linear algebra algorithms). Using this methodology, they report that the embedded FPU FPGA uses 58% fewer logic elements. Unfortunately, no details are given on the architectures of the benchmarks, but since there are relatively few FPUs available, we assume that these are processing element (PE)-based implementations. PE based implementations suffer from interface and control overhead required for recirculating data to the compute units. The performance of vector-based operations, common in linear algebra algorithms, would therefore be negatively impacted.

In [8] Ho, et.al. evaluate embedding FP cores into an FPGA architecture using simple function benchmarks, such as an FFT butterfly, a 4 tap FIR filter, and 3x3 matrix multiply. They use published data from the IBM Power PC to estimate the FPU size at 570 4-LUTs (the estimation is based on a 5-level pipeline stage datapath of the processor; the potentially simpler FPGA FP blocks are likely to require less area). Reported results show that the small number of embedded FPUs added to the fabric can bring up to 18X area savings compared to traditional implementations on the simplistic set of benchmark designs. Unfortunately, the analysis does not consider more realistic benchmarks that require hundreds of operators. The size of the FPU only allows the integration of a reduced number per device (even scaling to larger, current devices the FPU count would still be low) so unrolled structures such as long scalar products would have to be mapped to the same few units, in a PE fashion. This leads to throughput degradation due to the overhead of data recirculation. Moreover, their proposed unit may only issue one multiplication or one addition per clock cycle; this is suboptimal since the size of the unit would provide sufficient routing resources to simultaneously use the two operators.

Ho, et.al. update their work in [10, 11] where both single and double precision FPUs are proposed. They propose a hybrid FPGA architecture embedding coarse-grain blocks consisting of floating-point operators into the FPGA fabric. The number of coarse-grain blocks as well as the number of FP blocks within the unit are parametrizable. For comparison with this work, we will focus on the single precision results only. They propose a Xilinx XC2V3000 (28772 4-LUTs) device with 16 single precision FPUs, with each FPU containing two floating point multipliers and two floating point adders. Cost is given at 7% die area total, with each FPU equivalent to 122 LUTs. Scaling to the current mid-range 20nm devices containing in the region of about 700K 4-LUT equivalents, we would expect a total of 1600 single precision operators. Fixed point scaling follows this pattern - the 96 18x18 multipliers in the XC2V3000 would translate to 2400 multipliers in a current device- the Altera Arria 10 660 [12] has 3356 19x18 multipliers and the Xilinx Kintex Ultrascale 756 [13] contains 2592 27x18 multipliers. The improved size of the FPU would allow running large realistic benchmarks in today's devices. Several architectural limitations would impact system performance and utility: (a) lack of local interconnect between FPUs would not support cascade modes, (b) inability to simultaneously utilize the FP cores in the FPU, and (c) the floating point only nature of the design.

In [9], Chong and Parameswaran improve on Ho's work by designing, rather than estimating, the double precision FPU. Their FPU is configurable as one double precision floating point multiplier-adder pair, or two single precision pairs. Their design is only 74 4-LUTs in area, although they have to scale the aspect ratio to ensure enough routing resources for the interface. As a result, their FPU grows to 288 4-LUTs in area. The authors state that they hope that their multimode design will justify commercial use of FPGA FPUs. Although their results are better than Ho's, their proposal still suffers from the application specific nature of the FPU (although they also support some integer operations, standard fixed point DSP elements are still included in the FPGA), as well as only small number of the FPUs because of their routing requirements.

In [14] an alternate approach for enhancing floating point performance on FPGAs is presented. Rather than embedding dedicated FP units in the fabric, the work focuses on how to use the existing resources more efficiently. The approach focuses on fusing a cluster of multiple floating point operators together, by recognizing that: (a) two connected floating point operations will have a redundant normalize-denormalize pair, and (b) wider internal datapaths can compensate the accuracy impact of skipping some rounding stages. Results show that logic can be reduced by approximately 50%; in [15] it is shown that the approach is actually more accurate than single-precision IEEE-754 in 75% of the cases, using a Cholesky matrix decomposition example. Although this soft floating point method is more efficient, it still has significant latency if many operators are used in a datapath. There is still a considerable amount of logic used, and system speed for large designs is typically 50% lower than the speed of a DSP block.

The embedded FPU works acknowledge the application specific nature of their designs; "potential to waste significant silicon for non-floating-point applications" [2], "Dedicated FPUs are wasted resources for designs that do not make use of them" [10], and "However, if unutilized, embedded FPUs waste space on the FPGA die" [9]. Consequently, from a commercial perspective FPU units are only viable if their overhead is significantly small for designs not using them.

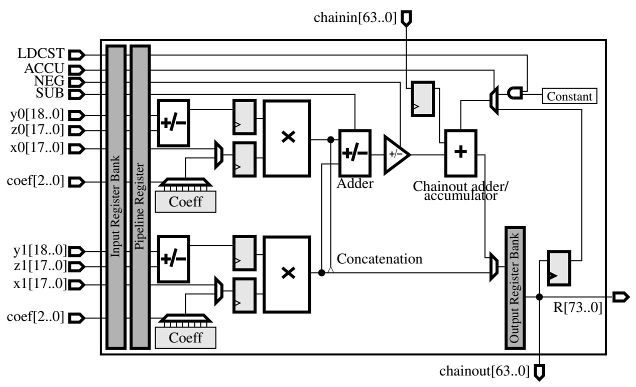
4. ARCHITECTURE

4.1. Why not another FPU?

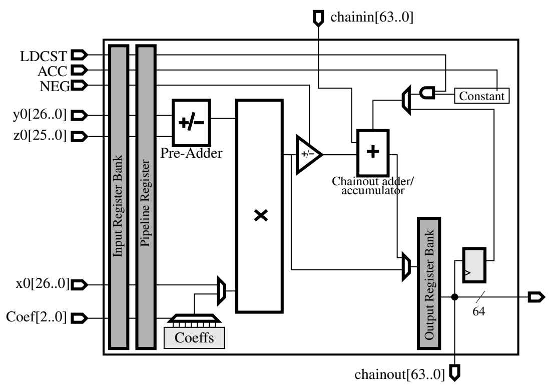
Many things need to be considered when including embedded floating-point features in an FPGA: (a) Size – routing density, routing congestion, and device redundancy, (b) Performance – the floating point function must run as fast as the maximum possible system speed, and in no way affect the performance of the existing fixed-point functions; (c) Utility – all fixed point functions must remain backwards compatible, or possibly align with a natural evolution of the DSP block, and (d) Tool support – the cost to support new synthesis and Place & Route tools, as well as the cost of IP migration all need to be taken into account. All of these underscore the need to include the floating-point features as part of the fixed-point DSP block, rather than another embedded core.

A good starting point is the StratixV DSP block[4]. Firstly, it already contains the hardware required for a single-precision mantissa multiplier under the form of a 27x27-bit fixed-point multiplier Fig. 1(b). Secondly, it has a good routing density with 108 data inputs which will support up to three 32 bit single precision inputs. With three inputs, the single-precision multiply-add operation can be supported within one block. Finally, it contains wide local interconnect paths that may be reused for cascade operations when configured in FP mode.

The routing density could be increased to support two or three double precision inputs, but this would have knock-on effects on routing congestion. The aspect ratio of the block could be changed to increase the number of inputs per block while simultaneously reducing routing density, but other device features such as row-based redundancy would not be supported. These considerations strongly suggest that a single precision multiply-add structure is a good starting point for an FPGA floating-point architecture. Even when using the embedded fixed point multipliers, a single precision floating point multiply and add pair would consume in the region of 700 LUTs and registers [16, 17].



(a) DSP block in dual 19x18 fixed-point multiplier mode



(b) DSP block in 27-bit fixed-point multiplier mode

Fig. 1. DSP block in two fixed-point configurations

4.2. Floating-point multiplier

The fixed-point configuration modes for DSP block are shown in Figure 1. The two 19x18 multipliers can be used independently, summed (both in Fig. 1(a)), or combined to create a 27x27 multiplier (Fig. 1(b)), which can be used to implement the mantissa multiplier for an IEEE-754 floating point multiplier. The IEEE-754 exponent calculation is simple, and the exception handling is a straightforward combinatorial wrapper around the arithmetic datapath. Because of the additional cost in silicon area and power, we decided not to support subnormal numbers as they are rarely used in FPGA arithmetic operators [17]. In fact, subnormal numbers are optional in the single-precision implementation of the OpenCL [18] standard which is used in a much wider range of device targets (such as GPUs and DSPs). We therefore flush to zero on the input and output of our floating point operators.

The greatest challenge is the implementation of the IEEE-754 round-to-nearest tie breaks to even (RNE) mode for the multiplier. Whereas a traditional implementation would require two carry-propagate-adders (CPA) on the critical path – one for the multiplier partial products and one for the rounding – a VLSI implementation of this structure would yield a large area and delay. In Section 5, we introduce some of the structures that make an efficient implementation possible such that overall, the inclusion of the FP multiplier has no impact on design performance, and only a 3% impact in DSP block area, including exponent and exception handling.

To complete the design of the floating point DSP block, a FP adder is also needed. Section 5 highlights some of the key implementation techniques allowing for a reduced implementation size while allowing for a one cycle execution at DSP nominal frequency.

The addition of both the multiplier and the adder has to be done with sufficient connectivity flexibility such that (i) scalar operations are available, (ii) multiply-add operations can be completed within the DSP block, and (iii) DSP blocks can be cascaded to easily and efficiently map operations such as scalar products onto them.

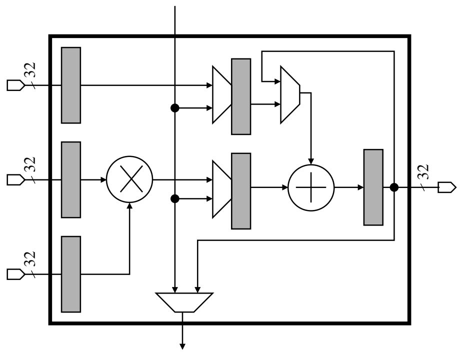


Fig. 2. DSP block in floating-point mode - simplified

4.3. Proposed DSP Block

Fig. 2 shows a logical block diagram of the connectivity (many of the registers used for performance or delay balancing have been omitted for clarity) supporting common arithmetic structures for both intra and inter block use. The latency for the DSP block in floating point mode is 2 (optionally 3 for performance) clocks for the FP multiplier and one additional clock when the adder follows the multiplier directly. An adder accessed independently of the multiplier will have a latency of 3 clocks. All or some pipeline stages can be optionally bypassed, but at the cost of a significant performance reduction.

Fig. 3 shows an alternate view of the DSP block, highlighting the balancing registers needed for the block to support all of the multi-row floating point modes at speed. All registers in the block are selectively by-passable, allowing any combination of multipliers and adders to be combined within a single block, or across two blocks, even though the maximum depth of the multiplier (three cycles) is different than the adder (combinatorial, with only an optional input and output pipeline). The adder can also be configured as a

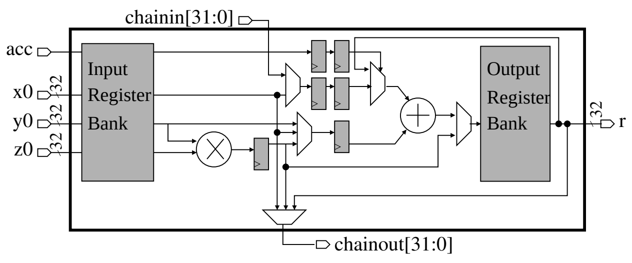


Fig. 3. DSP block in FP mode; balancing registers shown

floating point accumulator, enabled by the multiplexer immediately preceding the right hand side input.

The circuit design of the FP adder is critical for success. The fixed point DSP Block is highly optimized to fit into a row height (LAB/M20K height) in order to support redundancy. In a mid-range device there are almost 2000 DSP blocks, so the impact of the FP adder may be significant in terms of area, device architecture, and power consumption. The FP adder select inputs from a number of sources, including inside the DSP block, from outside the DSP block through general purpose routing, and from the adjacent DSP blocks though the chainin/chainout connection.

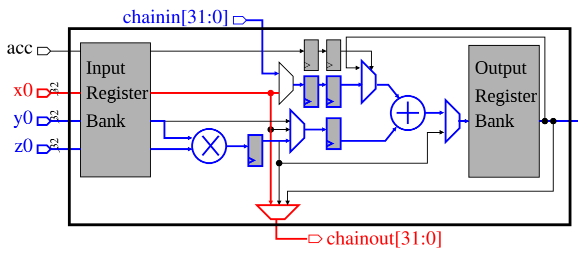
4.4. Dedicated vector structures

Vector structures are particularly useful for many unrolled DSP algorithms, such as FIR filters and matrix manipulations. New algorithms for common matrix decompositions such as Cholesky and QR have been introduced [15]. These algorithms work by reducing or eliminating data dependencies within rows and columns such that entire rows can be processed in parallel. The decomposition process is still an iterative process (though at a higher granularity) and low latency vector kernels are essential for enhanced performance. The multi-row DSP structure proposed in this work allows for a straightforward efficient logarithmic latency to kernel length ratio mapping.

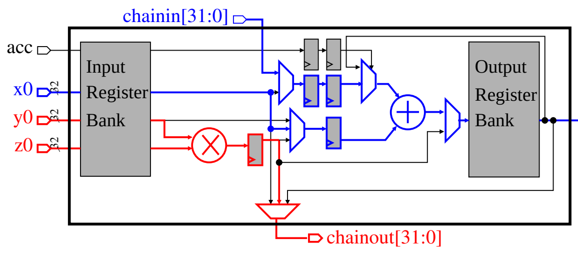
Fig. 4 illustrates two path configurations supported by the DSP block, which we will refer to as vector modes one and two, respectively. In vector mode one, the result of the floating point multiplication is directly fed to one input of the floating point adder, and the other input to the adder comes from the immediately adjacent block. In vector mode two, the multiplier is fed directly to the chainout connection (typically to the chainin connection of a DSP block configured as vector mode one); the two inputs to the adder come from outside the block, the left input through the input register bank, and the right input from the chainin connection.

The modes shown in Fig. 4 can be used to create a recursive tree structure of any size by using general purpose routing (but without soft logic). A logical block diagram (omitting many of the balancing register for illustrative simplicity) is shown in Fig. 6.

The basic building block is the addition of two products



(a) Vector One



(b) Vector Two

Fig. 4. DSP block in floating-point vector modes

– in this figure AB + CD (in the first two DSP blocks from the left) and EF + GH (in the next two DSP blocks). The AB+CD result is routed (via general purpose routing) into the third input of the second block from the left, and multiplexed into the left input of the floating point adder in that block. The EF+GH result is routed (again, via general purpose routing) to the third input of the third block from the left, and routed via the chainout/chainin connection to the second block from the left, and into the right input of the adder, producing AB+CD+EF+GH at the output of the DSP block. This process can be repeated to any degree to create any length of dot product. As there will be N FP adders and N FP multipliers in any collection of N DSP blocks, and only N-1 adders are required for a tree, there will be no function or connection shortage. While a floating adder inside a DSP block, i.e. following a multiply, has an additional latency of one clock cycle, an adder used for the recursive tree addition after the first level will add three clock cycles. Again counting from left block to right block, the first block is in vector mode one, the next block is in vector mode two, the next block in vector mode one, and the last two continuing in this alternating pattern. In general, this mode one/mode two pattern can be used to build any size reduction tree, although the connection length between two nodes in the later stages of the tree will tend to become very long.

For very large vectors, some of these general purpose routing connections may become the critical path, especially for a heavily utilized design. By inspection, the general purpose routing connections can be pipelined to any degree, as long as all connections at the same level of recursion are pipelined identically. Pipelining of the general purpose connections is also required if the length of the vector is not a power of two. The most efficient way to implement such a

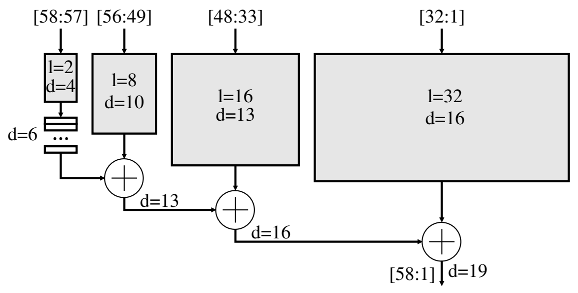


Fig. 5. Structure of a 58 Element Vector

vector would be to decompose the structure into increasing powers of two, balancing the latency between the first two sub-structures and adding them, and then balancing the delay of the addition output with the subsequent sub-structure before addition, and so on. For example, a vector length of 58 would be decomposed in the following substructure sizes: 2, 8, 16, and 32, with latencies of 4, 10, 13, and 16, respectively, and a total delay of 19 clocks. Fig. 5 shows this structure.

Supporting a an efficient mapping of this recursive computing structure is key for many of the application types, such as matrix decompositions.

5. ARITHMETIC MICRO-ARCHITECTURE

5.1. FP Multiplier

The greatest challenge is the implementation of the IEEE-754 round-to-nearest-even (RNE) mode, as the pedantic implementation consists of one CPA on the output of the multiplier, a 1 bit normalization circuit, and a final CPA for rounding. This is prohibitively expensive as the CPA is a significant portion of the multiplier in terms of both area and propagation delay, and a second adder would significantly affect both the fixed and floating point datapaths. The solution cannot be fully explained in space available in this paper. In summary, all possible rounded and normalized values for the output mantissa are calculated in the portion of the multiplier carry prefix structure corresponding to mantissa output position. These pre-calculated values are also used by the fixed point multiplier datapath; the carry prefix structures are distributed across the various fixed point decomposition points of the 74 bit CPA, and some of the larger fixed point additions are generated using carry select adders, choosing between some of the possible rounded floating point mantissa values for a certain range of the output. A detailed description of the FP multiplier mapping can be found in [19].

The only remaining multiplexer that may affect the fixed point performance is the final 2:1 multiplexer, selecting between fixed-point and the floating-point value post-exception handling. This will be a short path, as the exception handling conditions are largely calculated before the final output. Overall, the inclusion of the FP multiplier has no impact on design performance, and only a 3% impact in DSP block area, including exponent and exception handling.

5.2. FP Adder

We evaluated multiple FP adder architectures, finally choosing a dual path design. Like the FP multiplier, the RNE was the critical path, which we implemented by calculating all rounded possibilities of the mantissa before the final selection. The dual path FP adder is well known [20], preceding even the original IEEE-754 specification [21]. Many enhancements have been proposed since [22, 23], so we will not describe the architecture in detail beyond our RNE implementation.

The FP adder is logically and physically separate from the FP multiplier (the multiplier block also includes fixed point accumulators), so fused-multiply-add (FMA) is not supported, only multiply-add. We only needed to obtain a sufficient area and performance result; as the multiplier pipeline was much more complex in both gate numbers and functionality (supporting both floating-point, as well as many fixed-point modes), we decided to try a completely synthesizable approach to the adder, to allow a very malleable design during place and route. In common with many dual path architectures, the near path only implements subtraction; the far path also supports the near path addition. Consequently, the far path can have three ranges of values: $4 > x \geq 2$, $2 > x \geq 1$, and $1 > x \geq 0.5$. The rounding decision points for the three possible results can be calculated in parallel using three behaviourally described additions, with the actual result being selected based on the upper bits of the first adder ($x \geq 2$). The FP adder area is approximately 10% of the final DSP block core.

5.3. Integration and Verification

The base case DSP block and proposed additions were implemented in Verilog and synthesized with Design Compiler to a TSMC 28nm library. The functions were functionally verified using VCS to ensure no functionality was lost during the additions. Final area numbers are post-place-and-route, which was performed using Synopsys ICC and the design was DRC-clean. PrimeTime was used to verify that the block closes timing at 500MHz across standard process corners. The aspect ratio of the Block was determined by the Altera Arria 10 LAB height, as the DSP block was pitch matched to it, so that the row based redundancy could be applied to all features of the device. The DSP block is 2.5 times the width of the LAB.

We performed an extensive verification process, with one set of test vectors based on the methodology described in several different sources [24, 25] to verify a black-box view of the block. We also performed a white-box test, using

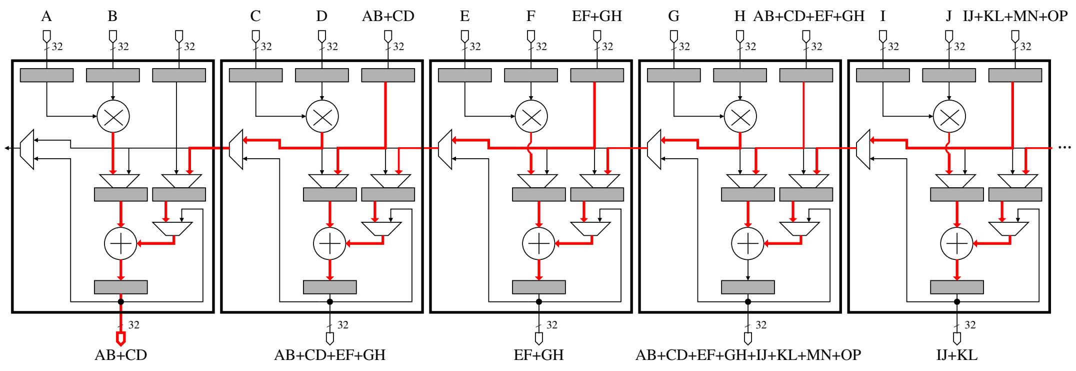


Fig. 6. DSP block in Recursive Vector Mode

a large set of internally generated vectors, which were designed to exercise areas of the architecture that we thought might be more prone to errors. For the multiplier, this was the overflow and underflow boundaries, as well as around the 1-bit normalization boundary. For the adder, this was around the cancellation boundary (where the subtraction of two numbers would result in a value close to, but not, zero), the near path/far path boundary, and the boundary where one operand would not affect the other (i.e. the one operand was much smaller than the other one). We have also successfully tested our implementation for conformance with the OpenCL standard, in both simulation, and hardware [26].

6. RESULTS

6.1. Application benchmarks

Methodology

We use a number of benchmark circuits to show the advantages of this new DSP block: a FIR filter, a FFT, a scalar product, a matrix-matrix multiply and a Cholesky decomposition. Each uses a different ratio of logic, memory and DSP blocks, a different DSP block to DSP connection pattern, and a different data flow pattern. All benchmarks use single-precision FP arithmetic, round to nearest (RNE) mode, with full support for infinities and NaNs but without subnormal support. Where possible, we compare the same design with and without FP DSP blocks enabled. Some of our example designs cannot be supported on any current FPGA without hard FP due to insufficient logic resources. Our soft FP implementations use the fused FP datapath synthesis supported by DSP Builder Advanced, and will give a more efficient soft logic only implementation than previous works.

For the FIR filter we present results for both a 1518 taps in order to highlight the top performance of the Arria 10 device, as well as a 128 taps FIR, which we use to show the resource improvements over a Stratix V device.

For the FFT we use two architectures. First, a 32K point FFT (32-way parallel) is performed in 1K cycles. The FFT uses radix 32, with each fully parallel 32-point FFT implemented using the split radix method. This architecture uses in excess of 1300 DSPs in FP configuration, and only fits in its current form on FP enabled Arria 10 devices. We report Flops performance using the conventional FFT flops metric which presumes that a size N FFT costs $5N\log_2(N)$ FP operations. Using this metric we obtain roughly $(5 \times 32K \times 15)/1K$ (FP operations/cycles). The second FFT architecture is a 64K point, 4-way parallel implementation consisting of 4-serial pipelines of depth 6 (each implemented as 3 radix 2^2 stages) feeding a 4-way parallel pipeline of depth 10 (implemented as 5 radix-4 stages) with a ROM implementation of the top-level twiddle factors. This architecture is used to highlight the advantages of using the embedded FP units as opposed to a soft floating point implementation (which is implemented on a Stratix V C2 device). The architectures have been generated using Altera DSP Builder Advanced Blockset 14.1 [27].

The scalar product family of tests include a 16-element dot product, and matrix-matrix multiply (MMM) of a 1280^2 matrix. We compare these architectures both with and without embedded FP support. The MMM uses a blocking decomposition of 128^2, with 8 separate scalar products of size 128. We also report an embedded FP only MMM chip-filling Arria 10 design, processing a 1600^2 matrix, with a blocking decomposition of 160^2, implemented using 8 scalar products of size 160. Both MMM benchmarks have been implemented using Altera OpenCL SDK 14.1 [28] and are fully functional architectures including all required communication interfaces.

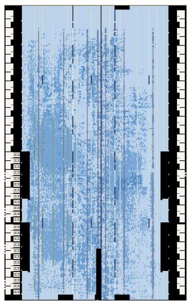
Finally, we benchmark a Cholesky matrix decomposition of size 254^2. The architecture used is similar to the previously presented works in [15, 29] and was implemented in DSP Builder Advanced (and is also part of the example designs). This benchmark compares the resource consumption of the Arria 10 and Stratix V implementations.

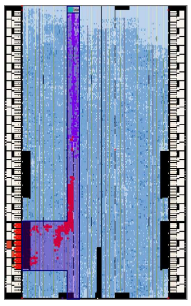
Discussion

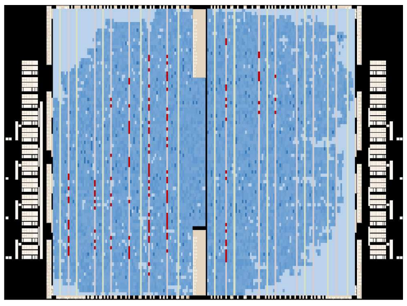
The benchmarks results are grouped in Table 1. The left column presents the benchmark, then subsequent columns present the resources and performance for designs without

Table 1. Performance comparison for embedded floating-point feature DSP block vs. traditional implementations

Algorithm	Perf. w/o FPDSP (Stratix V except MMM (1280) – Arria 10)	Perf. w FPDSP (Arria 10)
FIR 1518	no fit	470MHz (>1.4TFlops)
FIR 128	60881ALMs, 128 DSPs, 353 MHz	1676 ALMs, 131 DSPs, 365MHz
FFT 32K	no fit	78244 ALMs, 1364 DSPs, 356 M20K, 290MHz (≈700GFlops)
FFT 64K	92923 ALMs, 96 DSPs, 508M20K, 330MHz	13993 ALMs, 256 DSPs, 508 M20K, 360MHz
Vector (16)	7781 ALMs, 16 DSPs, 450MHz, 54 cycles	263 ALMs, 16 DSPs, 385MHz, 14 cycles
MMM (1600)	no fit	141442 ALMs, 1280 DSPs 1967 M20K, 285 MHz (730GFlops)
MMM (1280)	315061 ALMs, 1034 DSPs, 1732 M20K, 249MHz	61293 ALMs, 1034 DSPs, 1732 M20K, 280 MHz
Cholesky (254)	109914 ALMs, 260 DSPs, 333M20K, 275 MHz	12716 ALMs, 270 DSPs, 332 M20K, 277MHz


(a) 32K FFT on A10


(b) 1600 MM on A10


(c) 128-Tap SP FP FIR SV

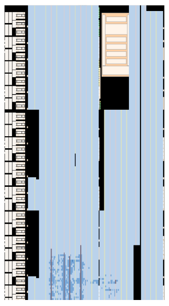

(d) 128-Tap SP FP FIR A10

Fig. 7. ChipPlanner views of mapped designs on Arria10 devices and corresponding StratixV devices where available

and with embedded FP support.

Firstly, the 1518 tap FIR filter shows the maximum attainable performance when only FP DSPs are used. The number of taps equals the number of DSPs of a large Arria 10 device (10AX115U5F45I3SP). The device has a layout similar to Fig. 7(a) and has 7 DSP columns of various length. The filter is therefore split into 7 corresponding sections with data delay chains running vertically. The jumps between columns span across the full height of the device and therefore require several stages of pipeline (we used 5 stages) to achieve the DSP block limited frequency of 470MHz, which translates to over 1.4TFlops.

The 128 tap FP FIR filter is presented in order to contrast the logic utilization reduction between a Stratix V device and the Arria 10 device. The logic reduction is more than 97% as virtually all of the logic in the filter is mapped to the DSP blocks. The number of DSP blocks increases from 128 to 131 in the hard FP case, as DSP Builder Advanced uses a default chunk size of 32 when splitting long dot-products. As a result, 3 additional adders are used to sum the 4 chunks together. Figures 7(c) and 7(d) visually present the device utilization ratio between the two architectures.

The 32K point FFT (32-way parallel) is used as an example of the capabilities of the new hard FP enabled devices. The number of DSPs used is close to the maximum count. Logic resources are used for constant (twiddle) multipliers and synchronization paths. The performance is close to 700 GFlops which is particularly interesting since FFTs have an imbalanced use of adders to multipliers. The logic utilization in the large device is less than 20%. Fig. 7(a) shows the relatively low logic resource utilization when FP DSP blocks are used. The smaller FFT demonstrates the implementation possibilities between hard and soft FP designs. Because of the non 1:1 multiplier:adder ratio in an FFT, the 85% logic savings are partially offset by a 2.66x increase in DSP blocks.

The next benchmark is a small 16-element real scalar. In this relatively trivial example, the logic savings of the FP DSP block approach 96% while also reducing latency by approximately 75%. A more complex MMM example provides a maximum performance of 730GFlops on multiplying square matrices of size 1600. The 8 vector products of size 160 used in the architecture are mapped to the 7 available DSP columns. Fig. 7(b) shows the device utilization for this design. A smaller example running on matrices of size 1280^2 and using 8 vector products of size 128 highlights the advantage of using the embedded FP DSP blocks. The non FP DSP design uses 315K ALMs, mapped to an Arria 10 device with no FP DSPs active for comparison. The logic savings in this case are close to 80%.

Finally, a DSP Builder Advanced Cholesky matrix decomposition example (part of the example degins) shows a push-button 88% improvement in logic resources over the equivalent Stratix V design.

6.2. Performance comparison with related works

The comparison of our results to the previous works is not straightforward. Some newer techniques have been introduced recently for FP datapath construction [14], which reduce the logic and latency of a dot product in the order of 50%, as well as new methods for matrix decompositions

[15]. The use of these methods would change the area and performance improvement analysis in the three previous papers. We also decided to normalize our results to the each of the three previous works individually. We believe that the area numbers previously reported do not completely correctly reflect the implementation in actual FPGA architectures, although we will not make an adjustment for this in our analysis for simplicity.

Beauchamp,et.al. [7] estimate the FPU area and performance from other published work. As the referenced work uses a more complex 5 stage pipeline, and is based on a commercial device, the FPGA scaling numbers are probably pessimistic; the integration overhead would be included in the reported area, and the 5 stage, higher performance, pipeline would likely be larger than the shorter FPGA embedded block.

Five benchmarks are used: matrix multiply, matrix-vector multiply, for product, FFT, and LU decomposition. No information on data sizes or algorithms, such as radix for the FFT, are given. There is no information on the architectures used – which will have a significant influence on the mapping to FPGA – for example, if dot product structures are used, or if a multiple PEs architecture is used. They state that division is mapped to logic, which may mean bit recurrence implementation, which will have a negative impact on performance with the associated long latency and routing stress.

Our methodology and results have been widely reported [15, 29] , including independent verification. We will compare dot product, FFT, and substitute a Cholesky decomposition for the LU decomposition. The Cholesky and LU algorithms are similar in complexity, and both use division. We will not attempt to normalize the comparison results – there is not enough implementation information given in [7].

Our logic reduction is typically 80%, in comparison to the 54% reported in [7]. This may be due to several reasons: a PE architecture will not scale as well as vector mapping, the FIR filter maps directly to our DSP blocks, and the recursive vector mode reduces overall routing congestion because of dedicated DSP to DSP connections. Newer algorithms for Cholesky decomposition allow long latency dot products to be used for matrix decompositions such as LU, Cholesky, and QR, because data dependencies are no longer the bottleneck. Beauchamp reports a 17% FPU device area – there are no details on how this was calculated, so we cannot normalize this to a current device – but in any case prohibitive for a mainstream FPGA. His estimates appear to require more area than the equivalent blocks implemented in [9] and [11].

Ho, et. al. [11] use a number of benchmarks. The three that are described in enough detail to implement are trivial - even in comparison to the contemporary FPGA used - matrix multiply, FIR, and FFT butterfly.

The matrix multiply is a 3 element vector, consisting of 3 multipliers and 2 adders. The FIR filter is a 4 tap FIR filter, assuming it consists of 4 multipliers and a binary tree of 3 adders, or perhaps it is of the alternate form, also requiring 4 multipliers, and 3 adders, with each adder summing the previous tap with the current tap. The FFT is a complex multiplier, followed by a complex adder. Ho gives an expected area for all of these in their Table IV(a), and reports an average of a 96% area reduction for these three functions. This is not a totally fair comparison, as their reported baseline numbers are for FP functions completely implemented in soft logic, with no use of the available embedded 18x18 multipliers.

These simple benchmarks can be directly mapped to our DSP blocks, using no soft logic other than registers needed to balance the datapath delays. We can implement the FIR filter directly using both forms: the direct form two structure with the following binary tree reduction is supported with the recursive vector mode of Fig. 4, and the direct form 1 structure is supported with the systolic mode connections, in a similar fashion to the existing fixed point FIR filter modes. Our implementation is more efficient from a whole chip perspective, as the FP functions are implemented in the existing DSP blocks, and DSP block to DSP block direct connections are available – and optimized for – common DSP structures, without using soft logic or routing.

Normalizing the reported FPU area into a current 700K LUT equivalent device would require 14% die area for 1600 DSP blocks, or approximately 4 times the cost of our design, but without the integer support. The difference is likely due to two reasons: the use of externally developed FP operators, which may not have been optimal for the area/performance requirements of this design, and the routing interface design, which would force a potentially difficult FPU aspect ratio.

Chong and Parameshawan in [9] built a configurable FPU structure that can be dynamically configured to support integer, single precision, and double precision arithmetic. They also built a double precision only FPU to understand the cost of providing multimode operation, which they reported as a 31% area and 21% delay increase. We believe that these numbers reasonably mirror that of the cost of configurability for commercial devices, although the incremental cost of adding single precision FP to our integer DSP blocks was in the range of 15% logic, without any performance reduction. Their FPU uses more complex double precision arithmetic, but the number of integer modes in a commercial device [12, 13] is very large, and not reflected in the modes supported in Chong. We disagree with their choice of a single path FP adder based on their stated reasons of design complexity and area, as the dual path architecture will typically synthesize to a much better area-performance result, which may impact the size of their FPU. Chong and Parameshawan use the same simple benchmarks as Ho, et. al. Their analysis shows a 68% reduction, rather than the 96% area reduction of Ho, because they include the embedded 18x18 multipliers in their calculations. Also, they use more soft logic around the FIR and matrix multiply benchmarks, presumably for data and coefficient storage. Again, we are able to map all of the arithmetic circuits directly to our DSP block, using only registers to balance delays, such as shown in our Fig. 5, or to store data and coefficients. Interestingly, normalizing their reported area results into a current 700K LUT device will require 17% for the equivalent of 1600 DSP blocks, although in this case, double precision is also supported.

In all three previous works, the system cost of the FPU blocks, normalized to current devices with DSP blocks supporting FP is similar in area; the interface density required seemingly the single most important influence.

Area is examined in all three previous works, but not routing optimization. In fact, all three designs are modified to allow full connectivity to the arithmetic structures.

7. CONCLUSION

We presented a new DSP block that can be configured to provide both and fixed and FP functions. Compared to earlier works, the proposed DSP block is truly integrated into the FPGA fabric, with high performance and efficient routing usage. It allows for a very high density of over 1600 single precision FP DSP blocks on a mid-range 20nm device, supporting up to 1.6 TFLOPs using only the newly embedded features. The proposed floating-point solution builds upon the existing fixed-point DSP block, and has lower routing requirements that the fixed-point modes. No degradation of fixed point performance or density occurs, and any mix of fixed and floating-point operations can be configured. In fact, this FP enhanced FPGA behaves just like a traditional FPGA architecture with fixed-point only capability, with essentially no cost or power impacts to non-floating point users.

8. ACKNOWLEDGEMENTS

We kindly thank Simon Finn and Tomasz Czajkowski for their contributions to application benchmarking.

9. REFERENCES

[1] F. de Dinechin, J. Detrey, I. Trestian, O. Creţ, and R. Tudoran, "When FPGAs are better at floating-point than microprocessors," ENS Lyon, Tech. Rep. ensl-00174627, 2007, http://prunel.ccsd.cnrs.fr/ensl-00174627.

[2] F. de Dinechin and B. Pasca, "Floating-point exponential functions for DSP-enabled FPGAs," in *FPT*. IEEE, 2010.

[3] *7 Series FPGAs Overview - Product Specification*, 2014. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[4] *StratixV Device Handbook*, 2011, http://www.altera.com/literature/hb/stratix-v/stratix5_handbook.pdf.

[5] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–58, 29 2008.

[6] E. Roesler and B. E. Nelson, "Novel optimizations for hardware floating-point units in a modern FPGA architecture," in *FPL'02*, London, UK, UK, 2002, pp. 637–646.

[7] M. Beauchamp, S. Hauck, K. Underwood, and K. Hemmert, "Architectural modifications to enhance the floating-point performance of FPGAs," *VLSI*, vol. 16, no. 2, pp. 177–187, Feb 2008.

[8] C. Ho, P.-W. Leong, W. Luk, S. J. E. Wilton, and S. Lopez-Buedo, "Virtual embedded blocks: A methodology for evaluating embedded elements in FPGAs," in *FCCM*, April 2006, pp. 35–44.

[9] Y. J. Chong and S. Parameswaran, "Configurable multimode embedded floating-point units for FPGAs," *VLSI*, vol. 19, no. 11, pp. 2033–2044, Nov 2011.

[10] C. H. Ho, C. W. Yu, P.-W. Leong, W. Luk, and S. J. E. Wilton, "Domain-specific hybrid FPGA: Architecture and floating point applications," in *FPL*, Aug 2007, pp. 196–201.

[11] C. H. Ho, C. W. Yu, P. Leong, W. Luk, and S. J. E. Wilton, "Floating-point FPGA: Architecture and modeling," *VLSI*, vol. 17, no. 12, pp. 1709–1718, Dec 2009.

[12] *Arria10 Device Overview*, 2014, http://www.altera.com/literature/hb/arria-10/a10_overview.pdf.

[13] *UltraScale Architecture and Product Overview - Advance Product Specification*, 2014, http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf.

[14] M. Langhammer, "Floating point datapath synthesis for FPGAs," in *FPL*, sept. 2008, pp. 355 –360.

[15] S. Demirsoy and M. Langhammer, "Cholesky decomposition using fused datapath synthesis," in *FPGA*, 2009, pp. 241–244.

[16] K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," in *FPGA*. ACM, 2004, pp. 171–180.

[17] *LogiCORE IP CORDIC v7.0*, 2013, http://www.xilinx.com/support/documentation/ip_documentation/floating_point/v7_0/pg060-floating-point.pdf.

[18] Khronos OpenCL Working Group, *The OpenCL Specification, version 1.2.19*, November 2012. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf

[19] M. Langhammer and B. Pasca, "Design and Implementation of an Embedded FPGA Floating Point DSP Block," Altera," Research Report, Dec. 2014. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01089172

[20] P. M. Farmwald, "On the design of high performance digital arithmetic units," Ph.D. dissertation, Stanford, CA, USA, 1981.

[21] "IEEE Standard for Binary Floating-Point Arithmetic," *ANSI/IEEE Standard, Std 754-1985,* New York, 1985.

[22] P.-M. Seidel and G. Even, "On the design of fast IEEE floating-point adders," in *ARITH*, 2001, pp. 184–194.

[23] ——, "Delay-optimized implementation of IEEE floating-point addition," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 97–113, Feb. 2004.

[24] J. T. Coonen, "Contributions to a proposed standard for binary floating-point arithmetic (computer arithmetic)," Ph.D. dissertation, 1984, aAI8512788.

[25] Z.-S. A. Liu, "Berkeley elementary function test suite," Department of EE and CS, UC at Berkeley, Tech. Rep., Dec. 1988.

[26] T. S. Czajkowski, "Silicon verification using high-level design tools," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, 2015.

[27] "DSP Builder Advanced Blockset," http://www.altera.com/technology/dsp/advanced-blockset/dsp-advanced-blockset.html.

[28] "Altera OpenCL SDK," http://www.altera.co.uk/products/software/opencl/opencl-index.html.

[29] "An Independent Analysis of Altera's FPGA Floating-point DSP Design Flow," 2011. [Online]. Available: http://www.bdti.com/MyBDTI/pubs/2012_Altera_FloatingPoint_Design.pdf